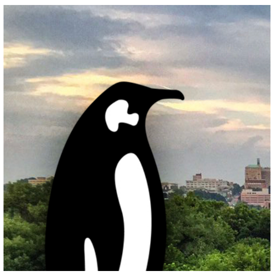

Contribution ID: **294** Type: **not specified**

Zero Copy Receive using io_uring

Wednesday 15 November 2023 09:30 (30 minutes)

Memory bandwidth is a bottleneck in many distributed services running at scale as I/O bandwidth has not kept up with CPU or NIC speeds over time. One limitation of kernel socket-based networking is that data is first copied into kernel memory via DMA, and then again into user memory, which adds pressure to overall memory bandwidth and comes with a CPU cost. The classic way of addressing this limitation is to bypass the kernel entirely, moving packets either to host memory (e.g. DPDK, PF_RING and AF_XDP sockets) or device memory (e.g. RDMA, RoCE and InfiniBand) without involving the kernel at all. These require re-implementing the networking stack either in userspace or the NIC hardware, and re-working applications which can no longer assume generic kernel-based socket I/O.

With hardware support for features such as flow steering and header splitting and new kernel features such as page pool memory providers, it is now possible to have a hybrid solution that sits in between full kernel bypass and full kernel copying. This paper proposes a way of doing zero copy network RX into host userspace memory using io_uring for the user facing API. We use header splitting to direct headers into the kernel for handling the stateful aspects of the networking layer, and payloads directly to its intended destination in userspace memory without copying using DMA. Flows are forwarded to specific RX queues configured for this using flow steering.

In this paper, we first discuss our design using io_uring from end to end in detail. We then compare our design with existing zero copy RX APIs in kernel and kernel bypass methods, and share some preliminary performance results after that. As part of this, we go into kernel features (both WIP and still in proposal) that are needed to support our design.

In the second part of the paper, we discuss the limitations of zero copy receive, and the challenges of fully making use of it in userspace applications. Unlike TX where data size is known ahead of time, RX data size is unpredictable and potentially bursty. To get the most out of zero copy receive, the write end has to coordinate closely with the receive end to agree on the exact shape of the data in order to avoid more copies down the line. For example, if the final destination is to write to a block device, then there are requirements for O_DIRECT to work such as alignment. Finally, we talk about our plans for further work e.g. extending support to GPU device memory.

Primary authors: WEI, David (Meta); BEGUNKOV, Pavel (Meta)

Presenters: WEI, David (Meta); BEGUNKOV, Pavel (Meta)

Session Classification: eBPF & Networking

Track Classification: eBPF & Networking Track